06

Reviendo la Composición de Objetos

Transcripción

[00:00] ¿Cómo lo voy a hacer? Aquí voy a tener un private AutenticacionUtil que después mi caja de herramientas. Es uno que está aquí abajo, vamos a llamarlo útil nada más, utilitario. Y en el constructor, cuando yo estoy creando mi cliente, vamos a definirle aquí un constructor.

[00:30] Tenemos uno por defecto y yo le voy a decir que útil, bueno, vamos a la buena práctica, this.util va a ser igual ¿a qué? A new AutenticacionUtil. Perfecto. Entonces yo estoy siempre inicializando por defecto AutenticacionUtil en cada ocasión. ¿Cuál es el beneficio de esto?

[01:05] Si se dan cuenta es una especie de composición de objetos. Es lo mismo que hemos hecho en el caso de la cuenta, por ejemplo aquí. Instanciamos un cliente y lo inicializamos como un cliente vacío. ¿Recuerdan en el curso anterior? Hicimos esto para que el cliente nunca inicie con null, pero esto era lo que definimos como composición de objetos.

[01:28] Un objeto dentro de otro, en el cual tiene completamente sentido y es inicializado automáticamente. De la misma forma estamos aplicando el concepto de composición de objetos aquí. ¿De qué forma? Aquí en útil tenemos una sola forma de inicializar sesión. ¿Cuál es? Primero setear la clave, segundo comparar la clave, una forma básica. Podemos añadir reglas de negocio? Sí se puede.

[02:00] Y tenemos un solo punto de entrada. ¿Cuál es la ganancia? La ganancia es que aquí en iniciarSesion yo simplemente puedo decirle return

this.util.iniciarSesion y le mando la clave. Y de esta forma, este código que había aquí ya pasa a una clase útil. Pero ustedes se preguntarán: ¿pero no es lo mismo? ¿No sería como hacer la misma cosa? Aquí también le puedo poner setClave(clave).

[02:34] Aquí, si lo aplicas a una sola clase, puede que sea lo mismo, puede que realmente no tenga el menor sentido. Pero recuerden que hay tres clases creo que inician sesión en este sistema. ¿Esas clases que inician sesión cuáles son? Son cliente, administrador y gerente. Entonces, allí me está dando un error, vamos a ver qué es. Es un punto y coma que me faltó aquí, perfecto. No lo vi. Listo.

[03:09] Y aquí en administrador vamos a hacer la misma jugada exactamente igual. Incluso si se dan cuenta, aquí en cliente yo ya puedo eliminar la clave. ¿Por qué? Porque nadie la está usando, nadie está usando esa variable clave, entonces con esto ¿qué estoy logrando? Desacoplar completamente la lógica de autenticación inicioSesion de mis objetos pues de negocio.

[03:45] Aquí el administrador es la misma historia. Yo puedo borrar esto de acá y decirle que AutenticacionUtil, perfecto, y aquí en el constructor vamos a añadir aquí public administrador. Listo. Y aquí yo le voy a decir que un this.util es igual a new AutenticacionUtil. Perfecto.

[04:36] Y de esta forma aquí yo solamente voy a llamar a un método que es this.util. No, acá no, perdón, este de aquí estaba bien. Vamos a llamar aquí this.util.setClave, y con la clave parámetro. De la misma forma acá, util.iniciarSesion, con la clave. Y si se dan cuenta, si bien implementamos la interfaz y estamos implementando los métodos, desacoplamos también completamente toda la lógica de autenticación.

[05:20] Todo está aquí. Entonces el código que habíamos acoplado, incluso la clave ya queda a cargo de AutenticacionUtil. Si se dan cuenta, un cliente y administrador ya no tienen nada que ver con la clave o algo relacionado a la

autenticación. Ya está todo limpio. Estos son todos los beneficios de la composición de objetos.

[05:44] Esto no es un concepto nuevo. Solamente es la aplicación de un concepto que ya habíamos aprendido a nuestro concepto actual en el que estamos mezclando la composición de objetos con lo aprendido en herencia y polimorfismo.